Axion

Security Assessment

October 23rd, 2020

For :
Axion

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Georgios Delkos @ CertiK
georgios.delkos@certik.io

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Axion
Description	An ERC20 token implementation with an inflation mechanism via staking and a penalty-based auction system.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. f2e654900f5023df3289426e0870d65efed06ea1 2. cfcc6d13abda5c748ee04c68cf784515b7508d16 3. 0c5b3c6dbfa2f3a24c5208cfa2e920fed3357788

Audit Summary

Delivery Date	October 23rd, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	October 19th, 2020 - October 23rd, 2020

Vulnerability Summary

Total Issues	70
Total Critical	0
Total Major	1
Total Medium	3
Total Minor	7
Total Informational	59

Executive Summary

During the process of our audit, we pinpointed several findings in all categories, many of which were mostly optimizational. The Axion team remediated all Major, Medium and Minor severity findings, however a lot of Informational findings that can greatly optimize the codebase and reduce the gas cost incurred by interacting with the contracts have been pointed out and should be taken into account for a next iteration of the codebase where applicable.

Files In Scope

ID	Contract	Location
AUC	Auction.sol	contracts/Auction.sol
BPD	BPD.sol	contracts/BPD.sol
FSP	ForeignSwap.sol	contracts/ForeignSwap.sol
IBP	IBPD.sol	contracts/interfaces/IBPD.sol
ITN	IToken.sol	contracts/interfaces/IToken.sol
IAN	IAuction.sol	contracts/interfaces/IAuction.sol
ISG	IStaking.sol	contracts/interfaces/IStaking.sol
IFS	IForeignSwap.sol	contracts/interfaces/IForeignSwap.sol
ISB	ISubBalances.sol	contracts/interfaces/ISubBalances.sol
NSP	NativeSwap.sol	contracts/NativeSwap.sol
STA	Staking.sol	contracts/Staking.sol
SBS	SubBalances.sol	contracts/SubBalances.sol
TOK	Token.sol	contracts/Token.sol

Findings

ID	Title	Type	Severity	Resolved
STA-01	Unlocked Compiler Version	Language Specific	Informational	
STA-02	Declaration Naming Convention	Coding Style	Informational	
STA-03	Contract-Level Tight-Packing	Gas Optimization	Informational	
STA-04	Redundant Variable Initialization	Coding Style	Informational	
STA-05	Variable Mutability Optimization	Gas Optimization	Informational	
STA-06	Inexistent Access Control	Control Flow	Minor	
STA-07	Assignment Location	Gas Optimization	Informational	
STA-08	Unconventional Function Name	Coding Style	Informational	
STA-09	Unoptimized `if-else` Conditionals	Gas Optimization	Informational	
STA-10	Unreachable `return` Statement	Gas Optimization	Informational	
STA-11	Code Duplication	Gas Optimization	Informational	
STA-12	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	
SBS-01	Visibility Specifiers Missing	Language Specific	Informational	
SBS-02	Struct Tight-Packing	Gas Optimization	Informational	
SBS-03	Variable Mutability Optimization	Gas Optimization	Informational	
SBS-04	Redundant `SafeMath` Utilization	Gas Optimization	Informational	
SBS-05	Utilization of Return Variable	Coding Style	Informational	
SBS-06	Variable Data Location Optimization	Gas Optimization	Informational	

ID	Title	Type	Severity	Resolved
SBS-07	Redundant Array Loop Assignment	Gas Optimization	Informational	ⓘ
SBS-08	Unoptimized `if-else` Conditionals	Gas Optimization	Informational	ⓘ
SBS-09	Conditional Optimization	Gas Optimization	Informational	ⓘ
SBS-10	Redundant `SafeMath` Utilization	Gas Optimization	Informational	ⓘ
SBS-11	Unreachable `return` Statement	Gas Optimization	Informational	ⓘ
SBS-12	Incorrect Error Message	Inconsistency	Informational	ⓘ
SBS-13	String Literal Representation	Compiler Error	Informational	ⓘ
SBS-14	Dangerous Conditional Execution	Volatile Code	Medium	✓
SBS-15	Redundant Statement	Gas Optimization	Informational	ⓘ
SBS-16	Redundant `SafeMath` Utilization	Gas Optimization	Informational	ⓘ
SBS-17	Unconventional Loop Logic	Gas Optimization	Informational	ⓘ
SBS-18	Multiple External Getter Calls	Gas Optimization	Informational	ⓘ
SBS-19	Unlocked Compiler Version	Language Specific	Informational	ⓘ
SBS-20	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	ⓘ
TOK-01	Mutability Specifiers Missing	Gas Optimization	Informational	ⓘ
TOK-02	Unsanitized Input	Logical Issue	Informational	ⓘ
TOK-03	Requisite Value of ERC-20 `transferFrom()` Call	Logical Issue	Minor	✓
TOK-04	Misleading `init` Function Prefix	Coding Style	Informational	ⓘ

ID	Title	Type	Severity	Resolved
TOK-05	Incorrect `require` Check	Logical Issue	Minor	✓
TOK-06	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	!
TOK-07	Unlocked Compiler Version	Language Specific	Informational	!
TOK-08	Declaration Naming Convention	Coding Style	Informational	!
NSP-01	Unlocked Compiler Version	Language Specific	Informational	!
NSP-02	Redundant Variable Initialization	Coding Style	Informational	!
NSP-03	`require` Order	Gas Optimization	Informational	!
NSP-04	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	!
NSP-05	Requisite Value of ERC-20 `transferFrom()` Call	Logical Issue	Minor	✓
NSP-06	Variable Mutability Optimization	Gas Optimization	Informational	!
NSP-07	Inexistent Access Control	Control Flow	Minor	!
FSP-01	Unlocked Compiler Version	Language Specific	Informational	!
FSP-02	Calculation Optimization	Gas Optimization	Informational	!
FSP-03	Variable Mutability Optimization	Gas Optimization	Informational	!
FSP-04	Amount Inaccuracy	Logical Issue	Medium	✓
FSP-05	Duplicate External Calls	Gas Optimization	Informational	!
FSP-06	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	!

ID	Title	Type	Severity	Resolved
BPD-01	Unlocked Compiler Version	Language Specific	Informational	
BPD-02	Calculation Remainder	Mathematical Operations	Minor	
BPD-03	Unconventional Logic	Gas Optimization	Informational	
BPD-04	Variable Mutability Optimization	Gas Optimization	Informational	
BPD-05	Utilization of Return Variable	Coding Style	Informational	
AUC-01	Unlocked Compiler Version	Language Specific	Informational	
AUC-02	Redundant Variable Initialization	Coding Style	Informational	
AUC-03	Redundant Conditional	Logical Issue	Medium	
AUC-04	Double Payout	Logical Issue	Major	
AUC-05	Conditional Optimization	Gas Optimization	Informational	
AUC-06	Declaration Naming Convention	Coding Style	Informational	
AUC-07	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	
AUC-08	Storage of `_msgSender()` to Memory	Gas Optimization	Informational	
AUC-09	Variable Mutability Optimization	Gas Optimization	Informational	
AUC-10	Inexistent Access Control	Control Flow	Minor	
AUC-11	Redundant Type-Casting	Coding Style	Informational	
AUC-12	Dead Code	Coding Style	Informational	

STA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Staking.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-02: Declaration Naming Convention

Type	Severity	Location
Coding Style	Informational	Staking.sol L51

Description:

The linked declarations do not conform to the Solidity style guide with regards to its naming convention. Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Recommendation:

We advise that the linked variable and function names are adjusted to properly conform to Solidity's naming convention.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-03: Contract-Level Tight-Packing

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L41, L51

Description:

The linked variables in sum occupy less than 256-bits whilst they exist in non-sequential order.

Recommendation:

We advise that either variable is moved right next to the other to optimize the gas cost of the contract as they would occupy a single storage slot instead of two separate ones.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Staking.sol L66

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-05: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L79-L86

Description:

The linked variable assignments are meant to be conducted once during the contract's initalization.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-06: Inexistent Access Control

Type	Severity	Location
Control Flow	Minor	Staking.sol L69-L87

Description:

The linked function that initializes the contract does not follow the access control convention of the other contracts whereby they declare a setter role that is revoked at the end of the `init` function's execution.

Recommendation:

We advise that the same access control paradigm is followed across all contracts.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-07: Assignment Location

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L362, L365

Description:

The linked assignment is executed in both cases of the `if` clause.

Recommendation:

We advise that it is instead moved outside the `if else` block.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-08: Unconventional Function Name

Type	Severity	Location
Coding Style	Informational	Staking.sol L414

Description:

The utility function `getNow0x` retrieves the current `block.timestamp` externally.

Recommendation:

This function, apart from being redundant as off-chain processes can easily retrieve the current `block.timestamp`, also utilizes an unconventional `0x` suffix. We advise that it is omitted.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-09: Unoptimized `if-else` Conditionals

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L302, L307-L308, L320

Description:

Each conditional beyond the first contains a redundant comparator of its respective preceding conditional.

Recommendation:

As each preceding case would guarantee the first comparison of each `else if` clause, it is possible to omit the first conditional of each linked condition and completely remove the last conditional rendering it a simple `else` statement.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-10: Unreachable `return` Statement

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L324

Description:

This statement will never be reached as the preceding `if` chain covers all cases of the function.

Recommendation:

We advise that it is omitted from the codebase.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-11: Code Duplication

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L97-L164

Description:

The linked functions contain the exact same statements apart from the value of one variable.

Recommendation:

We advise that they instead utilize a common `internal` or `private` function that accepts the specified variable as an input parameter, greatly optimizing the bytecode and gas cost of the contract.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

STA-12: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	Staking.sol L100, L137, L216

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-01: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	SubBalances.sol L62, L63

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-02: Struct Tight-Packing

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L26-L35

Description:

The `StakeSession` struct contains an unoptimized struct layout.

Recommendation:

Its layout can be optimized by re-ordering the `address staker` variable to instead exist after or before the `bool withdrawn` variable so that those two variables are tight-packed into the same storage slot.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-03: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L86-L92

Description:

The linked variable assignments are meant to be conducted once during the contract's initialization by the `SETTER_ROLE` address.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-04: Redundant `SafeMath` Utilization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L95

Description:

The linked statement conducts a `SafeMath` addition between the iterator variable `i` and the number literal `1`.

Recommendation:

This calculation will never overflow and as such, the utilization of `SafeMath` increases the gas cost of the function redundantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-05: Utilization of Return Variable

Type	Severity	Location
Coding Style	Informational	SubBalances.sol L128

Description:

The linked statement explicitly returns the return variable `shareAmount`.

Recommendation:

Instead of explicitly returning the variable, a `break` statement could be introduced here instead.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-06: Variable Data Location Optimization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L140

Description:

The linked variable is declared as `storage` yet all struct members are accessed.

Recommendation:

It is more optimal to instead store it as a `memory` variable as the lookup operations per struct member will be optimized.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-07: Redundant Array Loop Assignment

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L153-L156

Description:

The linked code segment retrieves the `stakeSession` from `storage` and assigns each of the 5 `payDayEligible` members to the `stakePayDays` array.

Recommendation:

It is possible to instead directly assign `stakeSessions[sessionId].payDayEligible` to `stakePayDays` as the arrays in question are statically-sized ones.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-08: Unoptimized `if-else` Conditionals

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L199, L204, L211

Description:

Each conditional beyond the first contains a redundant comparator of its respective preceding conditional.

Recommendation:

As each preceding case would guarantee the first comparison of each `else if` clause, it is possible to omit the first conditional of each linked condition and completely remove the last conditional rendering it a simple `else` statement.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-09: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L199

Description:

The equality case of the latter part of the comparison would yield the value equivalent of the clause's body.

Recommendation:

The linked `else if` clause can be optimized by making the latter part of the comparison a less-than-or-equal-to.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-10: Redundant `SafeMath` Utilization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L207

Description:

The linked statement conducts a `SafeMath` subtraction between the literal `714` and the value of `daysAfterStaking` i.e. `daysStaked.sub(stakingDays)`

Recommendation:

Both the subtraction of L206 as well as the subtraction of the linked line can be optimized by removing the redundant `sub` invocation as they are guaranteed to never underflow due to the `else if` conditional that precedes them.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-11: Unreachable `return` Statement

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L215

Description:

This statement will never be reached as the preceding `if` chain covers all cases of the function.

Recommendation:

We advise that it is omitted from the codebase.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-12: Incorrect Error Message

Type	Severity	Location
Inconsistency	Informational	SubBalances.sol L223

Description:

The linked error message says that the caller is not matching the sessionId, yet the `staker` member is evaluated in the conditional.

Recommendation:

We advise that the error message properly reflects the condition being evaluated.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-13: String Literal Representation

Type	Severity	Location
Compiler Error	Informational	SubBalances.sol L247, L298

Description:

The linked string literal for the error message utilizes single quotes (`''`) instead of double quotes (`""`).

Recommendation:

We advise that double quotes are utilized instead as single quotes are used for `byte` representations.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-14: Dangerous Conditional Execution

Type	Severity	Location
Volatile Code	Medium	SubBalances.sol L268-L277

Description:

The linked code block saves a specified `StakeSession` yet it is only executed when the duration of the staking is greater-than-or-equal-to the `basePeriod`, otherwise only shares are added to the `currentSharesTotalSupply`

Recommendation:

We advise that the `if` conditional is instead changed to a `require` check as we do not believe it is intended to add new shares to the total supply pool when a stake session is not created.

Alleviation:

After discussing with the Axion team, we came to the conclusion that a `require` check imposed here would halt the execution of external contracts interacting with the `SubBalances` contract and as such, it is more optimal to use an `if` conditional. The non-creation of a stake session during `callIncomeStakerTrigger` does not impact the soundness of the contract.

SBS-15: Redundant Statement

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L296

Description:

The linked statement does not affect the functionality of the code block.

Recommendation:

We advise that it is omitted.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-16: Redundant `SafeMath` Utilization

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L248, L299

Description:

The linked subtractions will never underflow yet utilize the `SafeMath` implementation.

Recommendation:

We advise that the `sub` invocations are replaced by literal subtractions (`-`) as the linked statements will never underflow due to the `require` statements that precede them

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-17: Unconventional Loop Logic

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L339-L355

Description:

The `generatePool` function is meant to iterate through the `subBalanceList` and attempt to create a pool out of the first `SubBalance` that is eligible

Recommendation:

We advise that the `return` statement from L352 is removed and an explicitly named `bool` return variable is utilized instead to allow multiple pools to be generated on a single run. Additionally, we advise the function's name to be changed to `generatePools`. The first change will lead to a lower total gas consumption if multiple big pay days are created in a single execution.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-18: Multiple External Getter Calls

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L370-L374

Description:

The linked statements conduct 4 external getter calls on the `foreignSwap` address.

Recommendation:

As the `ForeignSwap` implementation can be controlled, we advise that a single getter function is set on its implementation that returns all the necessary variables to greatly optimize the gas cost of the linked code block.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-19: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SubBalances.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

SBS-20: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	SubBalances.sol L144, L228, L232

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-01: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	Token.sol L17, L45

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-02: Unsanitized Input

Type	Severity	Location
Logical Issue	Informational	Token.sol L52-L54

Description:

The linked `for` block sets the `MINTER_ROLE` for each `address` contained in the `instances` array. However, no input sanitization takes place.

Recommendation:

The `instances[index]` address should be checked to not be equal to the zero address (`address(0)`) and optionally that it is not a duplicate value in the array as the `init` function can only be called once.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-03: Requisite Value of ERC-20 `transferFrom()` Call

Type	Severity	Location
Logical Issue	Minor	Token.sol L80-L83

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` function is safely invoked in all circumstances.

Alleviation:

After discussing with the Axion team, we concluded that the safe alternative of `transferFrom()` is not necessary here as the token implementation is meant to fully conform to the ERC20 standard so incompatibility with tokens such as USDT is of no concern.

TOK-04: Misleading `init` Function Prefix

Type	Severity	Location
Coding Style	Informational	Token.sol L79, L87, L93

Description:

The linked functions are meant to be called multiple times as their access control permits them to yet they are prefixed with the word `init` which would lead one to think those functions would only be called once.

Recommendation:

We advise that the `init` prefix is omitted from those functions as they are misleading with regards to their functionality.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-05: Incorrect `require` Check

Type	Severity	Location
Logical Issue	Minor	Token.sol L88

Description:

The linked `require` statement contains an incorrect error message and condition as it ensures that the amount to be withdrawn is greater-than-or-equal to the `swapTokenBalance` of the contract, which is incorrect as any value higher than that would cause the function to throw.

Recommendation:

We advise that the conditional is instead changed to a less-than-or-equal (`<=`) to comparator and that the error message's `<` symbol is swapped with `>`.

Alleviation:

The `require` check conditional was fixed to properly represent the error message it is accompanied by and function correctly.

TOK-06: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	Token.sol L97

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-07: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Token.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

TOK-08: Declaration Naming Convention

Type	Severity	Location
Coding Style	Informational	Token.sol L71

Description:

The linked declarations do not conform to the Solidity style guide with regards to its naming convention. Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Recommendation:

We advise that the linked variable and function names are adjusted to properly conform to Solidity's naming convention.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	NativeSwap.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-02: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	NativeSwap.sol L25

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-03: `require` Order

Type	Severity	Location
Gas Optimization	Informational	NativeSwap.sol L69

Description:

The linked `require` statement relies on a variable that is available earlier in the code.

Recommendation:

We advise that the `require` check is imposed as early as possible to avoid any redundant gas costs, i.e. right after the `amount` assignment.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-04: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	NativeSwap.sol L69

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-05: Requisite Value of ERC-20 `transferFrom()` Call

Type	Severity	Location
Logical Issue	Minor	NativeSwap.sol L46-L49

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` function is safely invoked in all circumstances.

Alleviation:

After discussing with the Axion team, we concluded that the safe alternative of `transferFrom()` is not necessary here as the token implementation is meant to fully conform to the ERC20 standard so incompatibility with tokens such as USDT is of no concern.

NSP-06: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	NativeSwap.sol L36-L41

Description:

The linked variable assignments are meant to be conducted once during the contract's initalization.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

NSP-07: Inexistent Access Control

Type	Severity	Location
Control Flow	Minor	NativeSwap.sol L28-L43

Description:

The linked function that initializes the contract does not follow the access control convention of the other contracts whereby they declare a setter role that is revoked at the end of the `init` function's execution.

Recommendation:

We advise that the same access control paradigm is followed across all contracts.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

FSP-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ForeignSwap.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

FSP-02: Calculation Optimization

Type	Severity	Location
Gas Optimization	Informational	ForeignSwap.sol L164-L165

Description:

The calculation of L165 is equal to `delta` minus a single `deltaPart`.

Recommendation:

We advise that it is replaced by `delta.sub(deltaPart)` optimizing its gas cost.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

FSP-03: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	ForeignSwap.sol L67-L77

Description:

The linked variable assignments are meant to be conducted once during the contract's initalization.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

FSP-04: Amount Inaccuracy

Type	Severity	Location
Logical Issue	Medium	ForeignSwap.sol L167-L168

Description:

The amount that is relayed to the `callIncomeDailyTokensTrigger` callback is higher than the actual minted amount.

Recommendation:

We advise that the same value is relayed to the callback as the callback internally updates the reserves of the token which will be inaccurate when this statement executes.

Alleviation:

The Axion team properly set the amount that the auction should be informed of, nullifying this exhibit.

FSP-05: Duplicate External Calls

Type	Severity	Location
Gas Optimization	Informational	ForeignSwap.sol L167-L173

Description:

The linked code segment contains two functions being invoked in sequence with different values.

Recommendation:

These values can instead be added to result in a single execution of those two function calls optimizing the gas cost of the function.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

FSP-06: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	ForeignSwap.sol L134, L151, L170

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

BPD-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BPD.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

BPD-02: Calculation Remainder

Type	Severity	Location
Mathematical Operations	Minor	BPD.sol L71-L76

Description:

The linked divisions and multiplications will result in a remainder that will forever be locked in the contract as it will remain unaccounted for since Solidity is prone to rounding errors.

Recommendation:

We advise that the last pool is instead assigned the result of the subtraction of the sum of the preceding pools from the full amount.

Alleviation:

The remainder issue was solved by retaining an additional variable called `remainderPart` that retains the remainder to be distributed to the final pool. We envision another optimization that can be made whereby instead of looping through all elements and conducting an `if` conditional, the loop iterates through all elements minus `1` and manually sets the final pool's amount equal to `remainderPart`. This would optimize the gas cost of the function.

BPD-03: Unconventional Logic

Type	Severity	Location
Gas Optimization	Informational	BPD.sol L79-L92

Description:

The linked code segment iterates through the `poolYearAmounts` array instead of directly retrieving the amount located at `poolNumber`.

Recommendation:

We advise that the `poolAmount` is used as an index directly and a `require` check, if necessary, or an `if` block precedes it to ensure the index is within bounds.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

BPD-04: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	BPD.sol L44

Description:

The linked variable assignments are meant to be conducted once during the contract's initalization.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

BPD-05: Utilization of Return Variable

Type	Severity	Location
Coding Style	Informational	BPD.sol L58

Description:

The linked statement explicitly returns the return variable `poolAmount`.

Recommendation:

Instead of explicitly returning the variable, a `break` statement could be introduced here instead.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Auction.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.12` which is specified in the project's `truffle-config.js` file, the contract should contain the following line:

```
pragma solidity 0.6.12;
```

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-02: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Auction.sol L81

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-03: Redundant Conditional

Type	Severity	Location
Logical Issue	Medium	Auction.sol L144

Description:

The first part of the conditional always yields `true` as the `index` variable is a `uint256` which is restricted to the non-negative range (`>= 0`). As such, the loop is equivalent to `while(true)` and the latter part of the conditional is not taken into account.

Recommendation:

The `||` joint of the conditional was meant to represent `&&` and as such, we advise that the conditional is adjusted to simply `points != 7`.

Alleviation:

The team applied this exhibit in full, omitting the former part of the conditional.

AUC-04: Double Payout

Type	Severity	Location
Logical Issue	Major	Auction.sol L241

Description:

The linked line sends double the `payout` to the `externalStake` function, in contrast to the intended bare `payout` as the referral system would be non-lucrative if this statement is intended.

Recommendation:

We advise that the addition of another `payout` is omitted from this line.

Alleviation:

The payout calculation was fixed by removing the invalid addition.

AUC-05: Conditional Optimization

Type	Severity	Location
Gas Optimization	Informational	Auction.sol L227, L315-L319

Description:

The `_calculatePayoutWithUniswap` function will either return a value that is less-than (`<`) `payout` or equal to it whilst the conditional of L227 checks a greater-than (`>`) condition.

Recommendation:

The condition can instead be changed to an inequality (`!=`) comparison which is more efficient gas-wise.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-06: Declaration Naming Convention

Type	Severity	Location
Coding Style	Informational	Auction.sol L111

Description:

The linked declarations do not conform to the Solidity style guide with regards to its naming convention. Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Recommendation:

We advise that the linked variable and function names are adjusted to properly conform to Solidity's naming convention.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-07: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	Auction.sol L217

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-08: Storage of `_msgSender()` to Memory

Type	Severity	Location
Gas Optimization	Informational	Auction.sol L172, L183, L185, L186, L190, L191, L192, L212, L215, L237, L243, L257, L264

Description:

The invocation of `_msgSender()` occurs repeatedly in the codebase.

Recommendation:

We advise that the result of its invocation is instead stored to an in-memory variable that is subsequently utilized in each statement.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-09: Variable Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	Auction.sol L101-L107

Description:

The linked variable assignments are meant to be conducted once during the contract's initalization.

Recommendation:

If all or some of those assignments are instead moved to the `constructor` of the contract, they can be greatly optimized by setting them as `immutable` thus reducing the gas cost involved in interacting with them significantly.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-10: Inexistent Access Control

Type	Severity	Location
Control Flow	Minor	Auction.sol L84-L109

Description:

The linked function that initializes the contract does not follow the access control convention of the other contracts whereby they declare a setter role that is revoked at the end of the `init` function's execution.

Recommendation:

We advise that the same access control paradigm is followed across all contracts.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-11: Redundant Type-Casting

Type	Severity	Location
Coding Style	Informational	Auction.sol L237

Description:

The linked `ref` variable is an `address` variable which is redundantly casted to an `address` variable.

Recommendation:

We advise that the type casting is removed.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

AUC-12: Dead Code

Type	Severity	Location
Coding Style	Informational	Auction.sol L52

Description:

The linked `prices` array is not utilized by the code of the `Auction` contract.

Recommendation:

As such, we advise its removal.

Alleviation:

The Axion development team has not provided a response to this exhibit yet.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.